THE GENERATION COMPONENT IN A MACHINE TRANSLATION SYSTEM

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

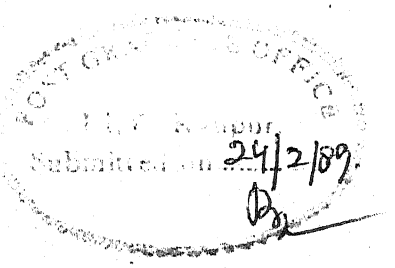
by

RIMLI SENGUPTA

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

FEBRUARY, 1989



CERTIFICATE

s is to certify that the thesis entitled THE GENERATION
IPONENT IN A MACHINE TRANSLATION SYSTEM by Rimli Sengupta
been completed under our supervision and that this work
not been submitted elsewhere for a degree.

r. P.R.K.Rao
ept. of Electrical Engg.

Dr. R. Sangal 24/2/89
Dept. of Computer Sc.

EE-1986-M-SEN-GEN

CONTENTS

chapter 1

INTRODUCTION

1.1 MOTIVATION

Since the inception of the digital computers, with their symbols as zeroes and ones, there has been a continual thrust on bridging the communication gap between human beings and the machines. Over the last two decades, significant steps have been taken in this direction, which includes the development of a large number of programming languages, based on Chomsky's context-free-grammers. These formal languages though, can be easily automated, are not compatible with human languages of communication. Till today, the human skills of understanding and expression of natural languages, are considered to be challenging problems in artificial intelligence.

The human ability to translate amongst different languages, is one of the very interesting problems in the area of natural language processing. All over the world, a large number of attempts have been made to mechanize the

process of translation. The machine translation project at I.I.T.(Kanpur)is one such attempt. The goal of the machine translation project is to achieve automated translation accross 14 different Indian languages.

There are mainly three different approaches to machine translation:

1. Direct approach: In this approach, for every pair of source and target language there is one dedicated translator. In the present context such an approach will need construction of large number of translators.
3. Intermediate approach: The intermediate approach is characterized by the intermediate language of representation. There exists a mapping from the source language to the interlingua and the interlingua to the target language.
2. Transfer approach: This approach lies somewhere between the above two approaches.

The machine translation project has adapted the intermediate appproach. With the intermediate approach one can achieve portability and modularity of design. In the worst case 14 different analyzers and 14 different synthesizers are needed. The intermediate language of representation is Conceptual Graphs (appendix 3).

The process of translation has to go through two different phases. In the fist phase, the analyzer, called the parser, which analyzes the source language text to produce the intermediate representation. In the second phase, the generator, generates from this intermediate representation a text in the target language. The problem of text generation appears to be particularly challenging, when one takes a careful look at the numerous complex processes involved.

The principal motive of this thesis is to explore the problem of text generation in the context of machine translation.

1.2 THE PROBLEM

The natural language generation involves a careful consideration of some issues like,

1. The style of representation in the text.
2. The relative ordering of the information by which the empahasis and the form of the text are related.
3. Syntactic constructions, conveying the mood of the text.
4. Surface structures which are important for complex and compound sentences.

5. Supressing the redundant information when it can be inferred.

Given the scenario that the analyzer part of the translator (viz, the parser) analyzes the source language text correctly, the information regarding the style, relative ordering of information, syntactic constructions etc., is correctly carried over from the source language to the intermediate language, and that the generator synthesizes the target language text from the output of the parser, the complexion of the problem changes somewhat. Since most of the traditional problems of language generation, as mentioned above, are solved by the analyzer and the relevant information is carried over to the intermediate language, the generator can directly utilize this information.

In the given context however, there are certain other problems which are more important. A given concept may have one or more than one realizations, in the target language. Such a situation can arise due to a several synonyms or closely related words in meanings. This problem of lexical choice has to be handled by the generator by making the most appropriate choice regarding the lexical entry.

Sometimes, the target language may have a special device for expressing a certain piece of information. Or the

form of the sentence under consideration may have to undergo certain changes. These idiosyncracies of the languages have to be carefully dealt with by the generator.

Another problem, very typical of this system is that, the parser and the generator are supposed to use the same data pool. Hence the generator should try to convert the data to it's required form. Without any additional burden on the language experts for creation of different data for the generator.

In a translation system, more often than not the data is likely to be incomplete due to its sheer size. To make the system robust enough to deal with this reality, the generator should exibit a graceful degradation, when dealing with incomplete data.

1.3 CONTRIBUTIONS

A generator has been designed and implemented that takes an intermediate repesentation as its input and generates the target language text. It deals with the problems outlined earlier in the ways described below:

(i) lexical choice :

It chooses words in the target language for a given concept if available from the concept dictionary(target language dependent). However, it can also make use of the information that a concept has a better lexical realisation either higher up (less specific) or lower down (more specific) in the concept type hierarchy. If more than one choices exist,the most appropriate choice is selected using symbiotic constraint satisfaction (chapter 3,section 5) ,ie, the lexical choices of the arguments of an action concept govern the lexical choice of the action concept itself and vice versa.

(ii) handling language idiosyncracy :

A transfer approach is taken to solve this problem. The intermediate representation passed to the generator is transformed to a form which is more compatible to the target language. A transfer specification syntax has been developed using which, the language expert can, with minimal effort, express the transfer rules for a language.

(iii) no extra onus on the languange expert :

An interface package has been developed which operates

on the data supplied by the language expert and generates all the data structures the generator requires. No separate data is needed to be prepared manually for the generator.

(iv) dealing with incomplete data:

This has been implemented for the data embodying the concept-type to lexical choice mapping. When a concept type has no lexical realisation in the concept dictionary, its type definition (Sowa, '84), containing less specific concepts, is invoked.

While designing the generator, the main aim was to maintain the structural layout of the parser while reversing the function of each component. That is, the relationship between the input and the output of a particular generator component is sought to be the same as that between the output and input of the corresponding parser component.

1.4 OUTLINE OF THE THESIS

Chapter 2 presents the Machine Translation system to provide the context in which the generator fits in.

Chapter 3 describes the transfer phase which performs both lexical choice and graph transfer.

Chapeter 4 talks about the rest of the phases of the generator.

Chapter 5 gives a limitations of this thesis and scope for future work.

Appendix 1 contains a few sample runs from the operational system . Appendix 2 contains input and output at every stage of the generator. Appendix 3 gives a brief introduction to conceptual graphs.

chapter 2

OVERVIEW OF THE MT SYSTEM

2.1 A BIRD'S EYE VIEW

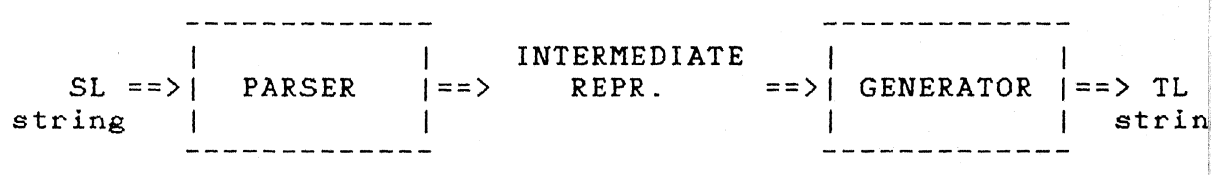


fig.2.1 THE MT SYSTEM
(SL= source language, TL= target language)

The approach adopted for translaton in the MT project at IITK involves usage of an intermediate representation, viz conceptual Graphs (CG). The source language text is analyzed by a parser which expresses the information content of the text in an intermediate form(CG), and from that, the target text is synthesized by a generator. The parser and the generator access language dependent data (which they share) and are themselves independent of the language to which they are applied.

2.2 WHY INTERLINGUA

Having an intermediate representation has many advantages. Firstly, with a direct approach of translation (ie, one translator for every language pair) across N languages, N*(N-1) differrent translators would be needed , whereas in this case one needs N parsers and N generators only. Further, since the programs depend on no specific language, one parser-generator pair with N blocks of data (one for each language) would suffice. This approach also has a very important by-product in that it lends powerful insight into the interesting devices that languages use to handle various natural situations. Questions as to how similar or dissimilar the devices are, why so, whether the similarities can be extended to a formal model of understanding etc. come up and are at least partially answered. An obvious disadvantage of having an intermediate representation is that it rules out adoption of any ad-hoc-ism for phenomena peculiar to a certain language pair - something which is frequently done and is perfectly feasible in the direct approach. But that's the price one pays for having a clean, language independent model.

2.3 WHY CONCEPTUAL GRAPHS

The notational "naturalness" of conceptual graphs makes it a very appealing method of knowledge representation for natural language processing. Also, since it has a direct mapping to first order logic with equality and modal logics (if embedded contexts are used), an inference mechanism can be incorporated with it. This would allow the usage of a knowledge base in disambiguating source language sentences in the MT system. It would also be possible to extend the usage of this system to contexts other than translation, viz, natural language understanding.

2.4 THE DATA PATH

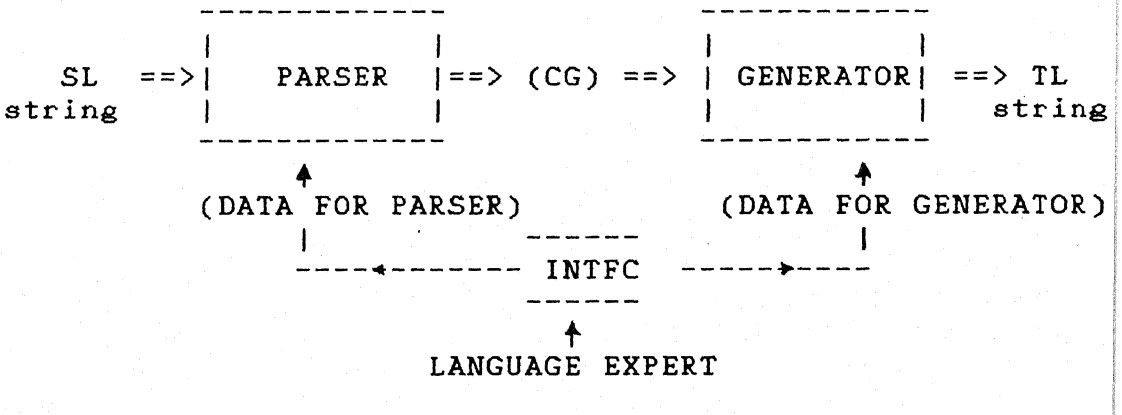


fig.2.2 THE DATA PATH

Interaction of the language expert with the MT system is shown in fig.2.2. S/he would be expected to provide all the relevant data about a language (say L1) and the INTFC block would generate the data structures for the parser and the generator (L1p and L1g) separately. The need to generate two sets of data arises largely from access efficiency considerations, since the two arms of the translator use the same data from different viewpoints. Therefore, the data used by the parser and the generator are identical in content but dissimilar in organisation. The basic premise that the usage of data is arm independent still holds.

For translating from L1 to L2 , the scheme in fig 2.3 would apply. In order to translate from L2 to L1, one would simply need to replace L1p by L2p and L2g with L1g.

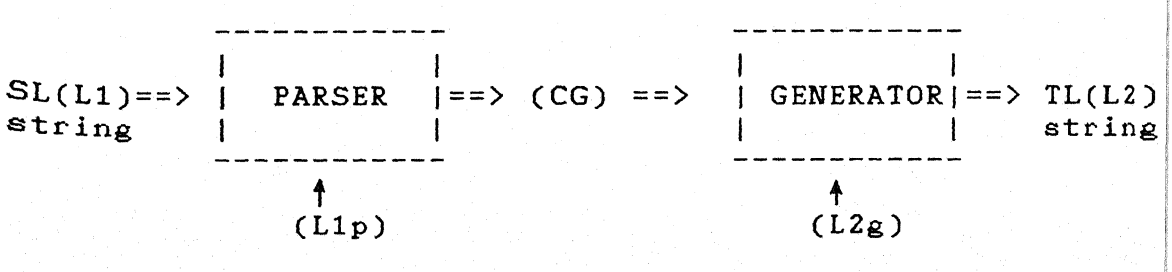


fig.2.3 TRANSLATING FROM
L1 to L2

2.5 A BRIEF LOOK AT THE PARSER

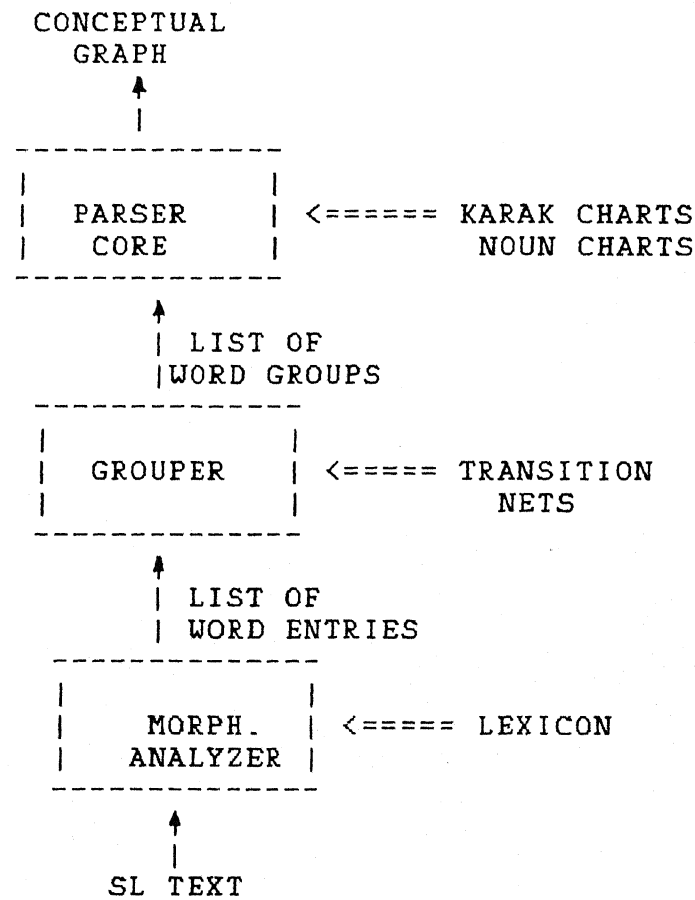


fig. 2.4 THE PARSER

2.5.1 The morphological analyzer

This block accesses the lexicon to look up all possible lexical entries for every word in the incoming text. Words are stored as tries for prefix sharing. Postfix sharing is

accomplished using end-tries. The corresponding piece of data for the generator is organised differently to facilitate efficient access of a word form given a lexical entry.

2.5.2 The grouper

This block uses transition nets to group lexical entries correctly (by using conditions on the arcs of the net) and returns a word group entry each time a group is successfully identified (by using actions at the states of the net). Implausible lexical entries for a word form are evidently eliminated by this block. Once again, the corresponding block in the generator, viz, the splitter does not use transition nets but a table lookup.

2.5.3 The parser

This block uses karak charts and noun charts (schema described in the next chapter) to inspect the selectional restrictions (yogyata) and demands(akankshya) of word groups and assigns semantic roles to them. Given more than one meaning of a word (expressed in a word group) it identifies

the correct sense by applying selectional constraints (specified in the schemata) on the potential source groups of a demand group. Disambiguating the sense is synonymous to identifying the concept a word group corresponds to. These concepts connected together with the relations (which are the identified roles) results in the final output, i.e., the conceptual graph. The corresponding block in the generator shares the schema completely but accesses them from a concept dictionary (3.1.1) containing the mapping of concepts to lexical entries for the target language.

2.6 ROUNDING UP

Often, devices for expressing a given situation vary across languages and taking care of such idiosyncracies may be accomplished by transforming the CG to a new one from which the closest target language sentence would be comfortably generated. For this end, a transfer component is used which also accesses language dependent information.

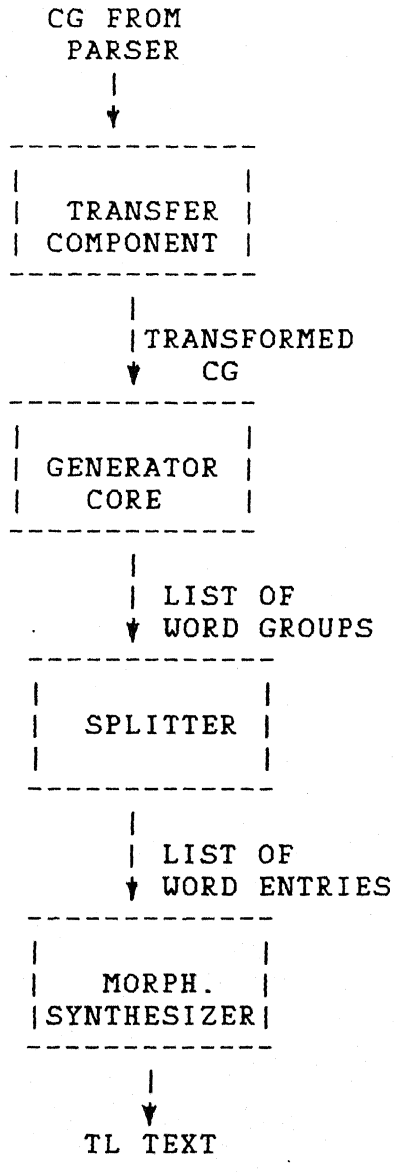


fig. 2.5 THE GENERATOR

It is not difficult to imagine the need for a similar block just after the parser-core for cases when most of the

languages have similar difficulty in expressing a given situation. Then, the parser's transfer component would affect the relevant change on the CG before passing it on to the generator. The final structure of the generator is shown in fig.2.5. Each of the components are described in detail in the forthcoming chapters.

chapter-3

THE TRANSFER COMPONENT

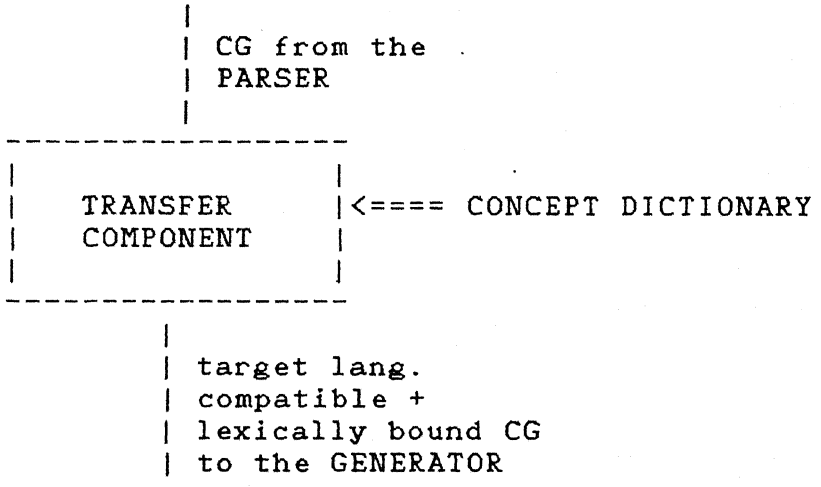


fig. THE XFER COMPONENT

This block plays a key role in language generation in the present context ,viz, machine translation. Its task is essentially twofold,

(a) to alter the input CG :

This is done in order to handle language idiosyncracies. Often, the target language(TL) in question has a special way of expressing the information embodied in

the input CG. In such cases, the closest(to human translation) TL sentence cannot be generated by operating the generator directly on the output of the parser. Our solution is to make use of the knowledge about the idiosyncracy and to transform the CG before handing it to the generator. A situation where such a transformation is needed is described below:

Supposing we are to translate from Hindi to Telugu and the sentence is,

```
rAma ne sadaka para eka GadI pAi
-------- ------------ --- ---- ---
 ram      on road      one watch found
```

(ram found a watch on the road)

The output of the parser looks like:

```
[pA-2]-
        (karta)->[manuRya:rAma]
        (karma)->[GadI-1]
        (desh-adhi)->[sadaka-2]
        (tam)->[yA].
```

The corresponding sentence in Telugu, expressing the same information, does not have an active construction.

So, the CG has to undergo a passivisation. This particular transformation is not desired for each and every instance of usage of the verb "find" but only when the fillers of the semantic roles of the verb satisfy certain conditions(3.1.4.1). The action(3.1.4.2) associated with the transformation here, involves altering the roles of certain arguments of the verb, viz,

karta--> sampradaan

karma--> karta

So, the output of the xfer component needs to be:

[pA-2]-

(sampradaan)->[manuRya:rAma]

(karta)->[GadI-1]

(desh-adhi)->[sadaka-2]

(tam)->[yA].

(b) to make a lexical decision about each of the concepts in the graph:

A concept may or may not have a lexical realisation in the target language. If it does, there may be one or more choices. In the latter case, the most appropriate choice needs to be identified. If no choice exists, the concept may be realised using either its type

definition(Sowa, '84) or another concept related to it by hierarchical dependency. As an example of the latter, supposing we are to translate from Bengali to Hindi and the sentence is:

rAma jala KAcCe
---- ---- -----
ram water is-drinking

(ram is drinking water)

The output of the parser is:

[KA-0]-

(tam)->[yA_rahA_hE]

(karta)->[manuRya:rAm]

(karma)->[pAnI].

The concept KA-0, which can be equated to the English "ingest", has no realisation in Hindi. However, KA-0 does have many subtypes in the concept type hierarchy for action concepts, one of which is pI-2 , whose selectonal restrictions on the arguments(viz, karta, karma etc.) is satisfied by the CG. So, the concept KA-0 in the CG is lexically bound to the choice corresponding to the concept type pI-2 in the concept dictionary(3.1).

3.1 CONCEPT DICTIONARY (CDict)

This is a language dependent database organised as follows:

key: concept type

value: (<lexical category> <lex. choices> <transfer rule> <lex. specification>)

3.1.1 DESCRIBING THE CONTENTS OF A CDICT ENTRY

(i) lexical category : Fillers for this slot are the usual categories, eg, noun, verb etc.

(ii) lexical choices : This feature takes a list of lexical entries (possibly empty), each of which are organised as a (<root> <schemata>) pair. The schema for action concepts are called karak charts(3.1.2) and those for the arguments(3.1.3) are called noun charts. Only pointers to the schema are stored here since more than one concept types often use the same schemata.

(iii) transfer rule : This slot is either empty (if no tranfer needed for the concept type) or contains a transfer

specification in a syntax described in 3.1.4.

(iv) lexical specification : This has the following possible values:

(a) nil -- no lexical specification

(b) a mark -- an atom = sub or super

(c) a designator denoting the presence of the type definition of the concept type.

Roles of (b) and (c) in making a lexical decision is elaborated in 3.1.5. Some sample CDict entries, are shown below:

Example 1. concept-type= uda-4; entry(telegu)=

```
(verb (vyApiMc uda-kch) nil nil)
```

This particular concept type has no lexical specification or transfer rule associated with it. Further, it has a single lexical realisation in Telugu.

Example 2. concept-type= suna-1; entry(telegu)=

```
(verb (vin suna-kch)
   ($case ([(not (%null kriya-vish))
            [(ch-con bola-1 )
             (ch-con tam 0_subjnc)
             (ch-rel karwA sampraxAna)]])) nil)
```

This one also has a single lexical realisation and has a

transfer rule associated with it. Section 3.1.4 contains an explanation of the rule.

Example 3. concept-type= KA-0; entry(hindi)=

(verb nil nil sub)

This concept-type has no realisation in hindi , but as the lexical specification field indicates, it may be realised via one of its subtypes in the concept type hierarchy(viz, pI-2).

3.1.2 THE KARAK CHART

This is a piece of data associated with every verb root in a language, embodying the meaning of the verb in some sense. It does so by providing detailed specifications or restrictions on the fillers of the semantic roles of the action it performs. It is a list of karak restrictions (one for each role eg, karta, karma etc) each of which is a structure of the following format:

(<karakname> <necessity> <vibhakti> <sf-expr>)

<karakname> : this is the name of a semantic role ,eg, karta.

<necessity> : this expresses how necessary the presence of

this karak is on the surface string or in the CG. The allowed values are: m(mandatory), d(desirable), o(optional), phi(should be absent).

<vibhakti> : this field contains the value of the vibhakti (possibly a list of values) taken up by the filler of this karak role, on the surface.

<sf-expr> : the value of this slot imposes selectional constraints on the filler of this karak role. This is used differently by the parser and the generator. The parser uses it to determine which karak role a word group satisfies and the generator uses it to ensure the semantic well-formedness of a CG.

A sample karak chart is shown below (for the verb jowa):

```
((karwA m 1 ($type prANI))
(karma m 2 (or($type prANI)($type yaMwra)($type sWAna)))
(karana d 3 (or($type prANI)($type yaMwra))))
```

The restriction for karana states that presence of such a karak filler is desirable, and if present, it must be a subtype of prANI or yaMwra.

3.1.3 THE NOUN CHART

This piece of data is associated with every noun root in a language, containing information about its inherent properties (viz, its meaning encoded as a concept type; super-species of that concept type ; material of which it is composed etc) and constraints on the fillers of the noun-noun relations (eg, sambandh , above, below etc) which may be attached to it. It is essentially a structure of the following description:

(<super-type> <concept-type> <upadhi> <concept-class>
<kriya-rels> <gen-rels> <dravya> <guna>)

3.1.4 TRANSFER SPECIFICATION SYNTAX

This is a tool offered to the language expert to capture the idiosyncracies of a language. The syntax is as follows:

```
($case  { (<condition>  [<action1> <actn2> ...])
          (             ........              )
                        .
                        .
                        .  })
```

which is a one-of case statement, i.e., if a <cond> is satisfied, each of the <actn>'s following it are performed on the graph and the statement is exited.

3.1.4.1 CONDITION

There are three condition primitives, viz, %type, %val, %null. A condition is either a primitive or a logical (and, or, not) combination thereof. When a transfer rule is bound to an action concept, the syntax described, has the power to access all its direct arguments, i.e., concepts connected to it via karaks, verb-verb relations etc. Similarly, for argument concepts, all its noun-noun relations may be accessed. examples:

(%type karma prANI) expresses the restriction that the karma of the action concept (to which the rule is bound) must be animate.

(%val karta <val1> <val2> ...) states that the karta must have a concept type which is one of the specified <val>'s.

(%null karaNa) returns true if in the given graph, the action concept has no karaNa relation attached to it.

3.1.4.2 ACTION

There are five primitive actions at present (subject to extension as and when concrete situations turn up) which have the power to handle all kinds of transfer acts in view, either alone or in combination. Combination is of course via adjacency only, i.e., actions are performed in the order specified.

Usage:

(a) (ch-con <rel> <con>) or (ch-con <con>)

<con> is a (<con-type> <con-ref>) pair for individual concepts and simply a <con-type> for generics.

Ex 1. (ch-con karma xahI) alters (in place) the type and referent of the karma of the action concept to xahI and *(generic marker) respectively. If the graph

was:

[jamA]-

(karwA)->[person:sIwA] ....(3.1)

(karma)->[xuXa].

with the transfer rule bound to the concept type "jamA", it becomes:

[jamA]-

(karwA)->[person:sIwA] ......(3.2)

(karma)->[xahI:*].

Ex 2. (ch-con KA-3) alters the type and referent of the action concept itself to KA-3 and * respectively. The graph in (3.2) would become:

[KA-3:*]-

(karwA)->[person:sIwA] ....(3.3)

(karma)->[xahI:*].

(b) (ch-rel <rel1> <rel2>)

Ex 1. (ch-rel karwA sampraxAna) changes (in place) the relation karwA (attached to the action concept) to sampraxAna. The graph in (3.3) would become:

[KA-3:*]-

(sampraxAna)->[person:sIwA]

(karma)->[xahI:*].

(c) (del <rel>)

Ex 1. (del karma) deletes the karma from the action concept. (3.3) would be altered to:

```
[KA-3:*]-
        (karwA)->[person:sIwA].
```

(d)(ch-link <rel> <con>)

Ex 1. (ch-link karma (dog kaiser)) changes the link between the karma relation and the karma argument to that between the karma relation and a concept whose type and referent are specified. Such a concept has to exist in the graph and the first such concept found , is used. If the graph is:

```
[xeKa]-
        (karwA)->[doctor]
        (karma)->[tongue]<-(sambandh)<-[dog:kaiser].
```

and if the rule is bound to xeKa, th new graph is :

```
[xeKa]-
        (karwA)->[doctor]                .....(3.4)
        (karma)->[dog:kaiser]->(sambandh)->[tongue].
```

(e) (add <con1> <rel> <con2>)

Ex 1. (add (man rAm) sambandh karma) would do the following to (3.4):

```
[xeKa]-
        (karwA)->[doctor]
        (karma)->[dog:kaiser]-
                                (sambandh)->[tongue]
                                (sambandh)<-[man:rAm].
```

(add karma sambandh (man rAm)) would have reversed the direction of the arrows.

Considering the sample graph:

```
[suna-1]-
        (karwA)->[man:rAm]
        (kriya-vish)->[loud]
        (tam)->[wA_hE].
```

the transfer rule (example 2, 3.1) bound to "suna-1"

```
($case ([(not (%null kriya-vish))
        [(ch-con bola-1)
         (ch-con tam 0_subjnc)
         (ch-rel karwA sampraxAna)] ]))
```

checks whether "suna-1" has a relation called "kriya-vish" in the graph. If it does, then the actions are performed one by one. The resultant graph is:

[bola-1:*]

(sampraxAna)->[man:rAm]

(kriya-vish)->[loud]

(tam)->[0_subjnc].

3.1.5 LEXICAL SPECIFICATION

If the lexical specification is a mark, it provides a relative change in strategy for lexical choice. Its presence implies that, for the target language in question, there exists a better (than the choices bound to itself) lexical realisation of the concept type in the lexical choices of the types either higher up (if mark = super) or lower down (mark = sub) in the concept type hierarchy. So, even if the field for lexical choices is non-empty, the type hierarchy is traversed first for a good choice, if nothing is available, only then is lexical choice field looked at. If there are no lexical choices either via the type hierarchy or in the the dictionary entry, or if none of the values in the dictionary entry apply, then the lexical specification is inspected once more to check whether the concept type has a type definition. If the definition is found , the

concept type is replaced by it(thereby expanding the graph) and the process of creating lexical bindings is continued for the new and hopefully realisable concepts in the expanded graph.

Situations employing the lexical specification field when it has a mark (sub or super) has already been illustrated earlier in this chapter. To motivate the usage of type definitions, one could imagine the MT system to be in a state where the intermediate language is sufficiently rich,(i.e., concepts are resolved into reasonably fine shades along with type definitions) and a new language is being installed in the system. Even while the data for the new language is incomplete(which will probably be the case), translation will not fail because any fine concept in the input (to the generator) CG will be expanded in terms of courser concepts which in turn will be lexically realisable. Also, when a concept is not realisable in a language(not because of incomplete data), using type definitions would be expedient.

For example, supposing we are to translate from Marathi to Hindi and the parser generates a CG which has

a concept "SrIKand" in it. Now, the concept dictionary for Hindi has the following entry for the concept type "SrIKand":

(noun nil nil def),

which states that this concept does not have any direct realisation, neither can it be realised by traversing the concept type hierarchy, but it does have a type definition available in the intermediate language, using which the generation procedure can continue.

3.2 IMPLEMENTATION

3.2.1 TRANSFER AND LEXICAL CHOICE MODULE

Procedure XFLC

purpose: to create lexical binding for a concept in a graph

input: a concept, its type, the graph where it occurs

output: a graph with the above concept lexically bound

algorithm:

XFLC (con cty gr)

1. Lookup CDict entry for cty
2. If <xf-rule> is empty, go to 3.

   else

apply rule on gr

if con-type of con has been changed due to xfer

go to 1. with xferred graph and cty = new con-type

lex-spec> is empty, go to 4.

if its a mark, follow it

i.e., if mark=sub, invoke XFLC for each subtype of cty until a choice is made.

ect <lex-choices>

if it is empty then,

.1.1 if <lex-spec> has type definition for con-type of con then

expand gr on con and go to 1.

else, FAIL.

if it has one value, ACCEPT.

else

try to get the appropriate value by constraint satisfaction

4.3.1 if successful, ACCEPT

else, go to 4.1.1

fig. 3.2 creating lexical binding for concept con in graph gr.

each concept (say with type c) in the graph, the

ocedure (xfer + lex.choice or XFLC) is performed

(called with cty = c). Firstly, the CDict entry for c is looked up, if the field for xfer rule is non empty, it is applied on the graph. if this action modofies c itself, then the procedure described so far is repeated until the xfer rule field is found to be empty. At this stage, the lexical specification is looked at. If its a mark then the type hierarchy is invoked with it, i.e.,if mark = super then this procedure is called with the every supertype of c until a choice is found. If a choice is found, the process is exited, else, the field for lexical choices in the CDict entry for c is inspected. If the field for lexical choices has one value (<root> <schema>) pair) then that is taken to be the choice and the process is exited. If it has more than one value, inappropriate choices are eliminated using a constraint satisfaction scheme described in the next section. In case none of the values in the lexical choice field are found to be appropriate or if it is empty to start with, the lexical specification field is looked up once more to check if the type definition for the concept (to be bound) type is available. If it is, the graph is expanded on the concept and lexical bindings are created for the new concepts. XFLC admits failure iff none of the above cases apply.

3.2.2 TRAVERSING THE GRAPH AND CONSTRAINT SATISFACTION

XFLC is called with the main action concept, its type and the graph. Through any route, the control has to come to 4.2 for creating a lexical choice. At this point, the status is that the action concept has one or more lexical realisations in the target language. Now, each of the concepts connected to the action concept by a relation are inspected. If the relation is a karak (eg, karta) , XFLC is applied to the argument concept and if its a verb-verb relation (eg, ktwa,shatri etc), the argument concept is pushed onto a stack (*S*) global to the procedure of traversal. If the action concept and the argument concept both have single realisations, lexical binding is created straightaway. For other cases ,viz, the action concept has one realisation and the argument concept has more than one realisaton or, the action concept has more than one realisation,conformity checking needs to be done for eliminating inappropriate choices. This is done in the following manner:

(a) For a lexical realisation of the action concept, its karak chart is invoked,

(b) For creating bindings for an argument concept (say, the karta) the relevant karak restriction is extracted from the karak chart,

(c) For a realisation of the argument concept, its noun chart is invoked,

(d) The sf-expression in the extracted restriction is applied on the noun chart,

(e) If (d) succeeds, the choice made in (c) is the correct choice for this argument concept and control goes to (b) with the next argument concept. If (d) fails, control goes to (c) with the next realisation of this argument concept.

(f) If the lexical bindings are successfully (i.e., (d) succeeds) created for each karak argument connected to the action concept, the lexical choice made for this action concept in (a) is the correct choice. If any karak argument fails to be lexically bound, control goes to (a) with the next realisation of this action concept.

Once the lexical choice of an action concept is successfully completed, i.e., all its karak arguments are lexically bound also, *S* is inspected. If its empty, the

transfer component is through, else, *S* is popped and the above procedure is repeated for the popped concept. If there is failure at any stage, i.e., the conformity check fails after the graph has undergone xfer, one step back-tracking is done. That is, the graph in its preceding stage of transfer is saved and in case of failure in the current stage, operation is resumed with the preceding stage. Since a verb does not exert any influence on its verb-verb relations, the previous steps of transfer don't need to be undone. As an example, one could think of a situation where the main verb and its immediate arguments are lexically bound and the graph has possibly undergone some transformation (say it is in state S1). Now, say the type hierarchy is being traversed for the choice of an action concept in *S* (i.e., one of the concepts bound to the main verb by a verb-verb relation) and mark= super. Supposing the current concept type could not yield a choice, but did transfer the graph on the basis of the <xf-rule> associated with it (say, R). Now, the next supertype of the action concept has to be considered and the rule associated with it is applied on the graph in its preceding state (viz, S1) of transfer ,i.e., before R was applied.

3.2.3 TRAVERSING THE TYPE HIERARCHY

In case the <lex-spec> for a concept type is a mark (as defined earlier) , type hierarchy is traversed. For each concept (action or argument), while the lexical binding is yet to be created and the type hierarchy is being traversed, all its supertypes, subtypes and types-visited-so-far are kept track of. These data structures are liquidated once a new concept needs to be lexically bound. They are initially empty (remain empty if the type hierarchy is not invoked) and are filled up in the following manner:

There are four queues, global to the process of lexical binding for a given concept (action or argument), for maintaining its supertypes and subtypes, viz, *NSB* (arg.,sub), *NSP* (arg.,super), *VSB* (actn.,sub) , *VSP* (actn.,super). Say, lexical bindings are being created for an argument concept and mark=sub. Then all the subtypes of this concept is inserted in *NSB* (if mark was = super, *NSP* would be used); for each of the concept types (cty) in *NSB*, XFLC is called. If the <lex-spec> is a mark and sub once more in the process of type hierarchy traversal,

then the subtypes of that cty is joined to the tail of *NSB* to maintain a breadth first ordering of traversal.

Consider the following situation: supposing the concept being bound has

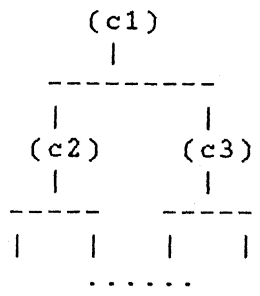


fig. Section of a dummy type hierarchy

type c2 and <lex-spec> = super and let the <lex-spec> of c1 = sub (i.e., the desired lexical choice lies with c3)- there is a possibility of looping indefinitely unless one keeps track of the types visited-so-far in the type hierarchy. So, each time a (new) cty is visited, it is recorded in a list and when the control shifts to another cty, if the membership test of that cty in the list succeeds, then it is not visited, else, it is recorded in the list and visited. The scheme described above for type hierarchy traversal is completely applicable to action concepts.

chapter 4

THE GENERATOR

This is the block which generates the target language sentence from the conceptual graph passed down by the transfer component. Conceptually, it can be segmented into three parts, essentially isomorphic with the parser but converse in functionality.

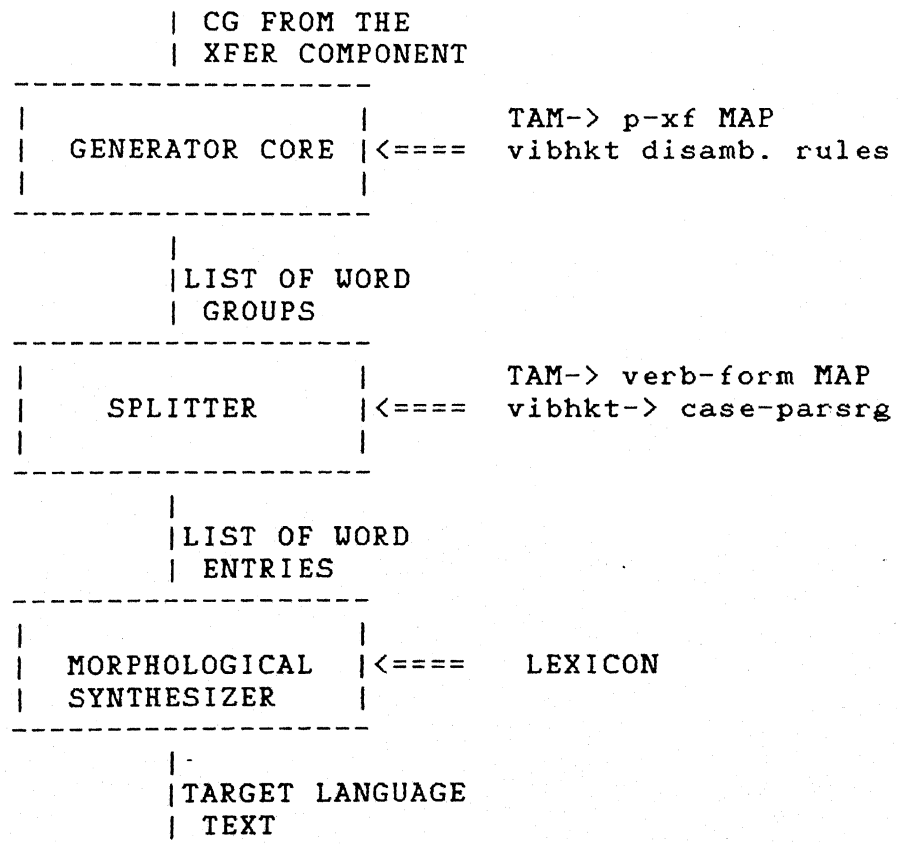


fig. THE GENERATOR

4.1 GENERATOR CORE

4.1.1 DATA ACCESSED

(A)TAM table : Tense, aspect and modality are three features whose composite values map onto the conjugated forms of a verb as uniquely as desired. In other words, they are variables whose domains are large enough to distinguish one verb form from the next, in a conjugated system. Each conjugated form is tagged by a tense-aspect-modality (TAM) marker, such that, given the verb root and the TAM, it is possible to retrieve the corresponding conjugated form of the root. Functionally, the TAM identifies a specific application of a verb. The TAM table is a database indexed on TAM tags containing:

(a) verb form : this is an expression which has the following format:

(<form of the root verb> (<auxilliary root> <form>)
(<aux. root> <form>).....)

if there are no auxilliaries for a TAM, the verb form is an atom which is the form of the root verb.

ex., (duratv (un past)) this implies that the root verb will be in durative form and the auxilliary verb has the root "un" and will be in past form.

(b) parsarg transfer rule (p-xf): in the TAM table, only the rule label is stored, the actual rule is a lisp procedure which accepts a karak chart and outputs another in which some of the karak restrictions have possibly undergone a change of content in the vibhkt and necy fields. The p-xf generates the application karak chart (TAM identifies an application, hence associated with it) from the language karak chart accessible from a lexical entry (3.1.1.(ii)).

The generator makes use of the p-xf rule label only, the TAM table is accessed once more by the splitter for the verb form.

(B)vibhakti disambiguation rules: If a karak restriction has a list of values in the vibhakti field, some global (to the target language) rules are used to extract the appropriate vibhakti. This requires world knowledge about

the concept which is the filler of the karak restriction. Thus the concept type hierarchy needs to be invoked here. The rules are organised as follows:

key: karak

val: set of rules, arranged as clauses so as to eliminate the non-applicable values.

eg., key = karta

val = [(($type manuRya) 1a) (($type prANI) 1b)]
this states that if the filler for the karta role is a subtype of manuRya then the vibhakti to be taken up is "1a", if the filler concept is not a subtype of manuRya but a subtype of prANI, then "1b" is to be used.

4.1.2 A TRIP WITH THE GENERATOR

(A) A list of phrase groups are created at first from the CG passed down by the xfer component("ng" and "vg" are abbreviations for noun-group and verb-group respectively).

Suppose, the graph passed down by the xfer component is :

```
[bETa-4]-
        (karwA)->[jUwA-1]-
                          (mod-adj)->[yah]->(pos)->[num:1]
                          (pos)->[num:2],
         (xeSa-aXi)->[pEra]-
                          (mod-sam)<-[manuRya:rAm]->(pos)->[num:3]
                          (pos)->[num:4],
          (kriya-vish)->[Tik]->(pos)->[num:5]
          (pos)->[num:6]
          (tam)->[0_sakawA_hE].
```

Lexical choices have been made for the concepts. Using them, the following structure is generated (target = bengali):

```
[(vg (root ha) (positn 6 )..... (tam 0_sakawA_hE) ..
                    ((kriya-vish ((adv Tik 5 ..)) )) )
  (ng (root juwo) (positn 2) ..... ((krkrel (karwA-of **))
                    (mod-adj ((adj ei 1 ..)) )))
  (ng (root pA) (positn 4) ..... ((krkrel (xeSa-aXi-of **))
          (mod-sam ((ng (root rAm) (positn 3) ....)))) )]
```

(B) After this, the karak chart for the root "ha" (carried in its entry shown above) is looked up. Then, using the tam (0_sakawA_hE, in this case) the TAM table is looked up for getting the p-xf rule (normal in this case

so no xfer on karak chart),the rule is applied and the transformed karak chart is used for vibhkt assignment. If for any restriction, >1 vibhakti's are present, vibhkt disambiguation rules are invoked. For this example, the restriction for xeSa-aXi looks like:

```
(xeSa-aXi m (7b 6) (or ($type manuRya) ($type prANI-anga)))
```

The disambiguation. rule for xeSa-aXi is looked up and that states : if the filler of this role is human, vibhakti=6. Since that fails, the other value is taken up. Now , the structure looks like :

```
[(vg (root ha) (positn 6) ..... (tam 0_sakawA_hE) .... ((... )) )
 (ng (root juwo) (positn 2) .. (vibhkt 1a).. ((krkrel (....))
                          (mod-adj ((...)) )))
 (ng (root pA) (positn 4) ... (vibhkt 7b .. ((krkrel (...))
                          (mod-sam ((...)) )))]
```

(C) Next, each word group is brought to the surface:

```
[(vg (root ha) (positn 6) .... (tam 0_sakawA_hE)..)
 (adv (root Tik) (positn 5) .... ((mod-adv-of **)) )
 (ng (root juwo) (positn 2) ... (vibhkt 1a).. ((krkrel (karwA-of **))))
 (adj (root ei) (positn 1) ... ((mod-adj-of **)))
 (ng (root pA) (positn 4) ... (vibhkt 7b).. ((krkrel (xeSa-aXi-of **))))
 (ng (root rAm) (positn 3) ......   ((mod-sam-of **)))]
```

(D) Vibhakti's are put for the noun-noun relations like mod-sam, above, below etc. They are usually fixed for a target language. So, for any such relation R, in a noun group (ng) if there is a value for the attribute R-of, the appropriate vibhakti for R is put. For mod-sam, 6 is put in the ng's containing a value for mod-sam-of,i.e., the one with root = rAm.

(E) Finally, the word groups are sorted according to the surface ordering specified in the CG. For cases when this default (corr. to the source text) ordering is not applicable , the transfer component imposes a new order on the CG on the basis of specified rules. The word groups are sorted here since splitting word groups creates new entries which are order sensitive. So the output of the generator is the following list of word groups (parse structure):

```
[(adj (root ei) (positn 1) .... ((mod-adj-of *)) )
 (ng (root juwo) (positn 2) .. (vibhkt 1a).. ((krkrel (karwA-of *))))
 (ng (root rAm) (positn 3) ... (vibhkt 6) ... ((mod-sam-of *)))
 (ng (root pA) (positn 4) ...  (vibhkt 7b)... ((krkrel (xeSa-aXi-of *))))
 (adv (root Tik) (positn 5) ... ((mod-adv-of *)))
 (vg (root ha) (positn 6) .... (tam 0_sakawA_hE)...) ]
```

4.2 SPLITTER

This block accepts the parse structure and generates a list of word entries.

4.2.1 DATA ACCESSED

(A) TAM->verb-form mapping : 4.1.1 (A)

(B) vibhakti-> case-parsrg mapping : vibhakti is a language independent notion whereas case and parsarg are features which have language dependent values that together expresses the vibhakti. A section of the map for bengali is shown below.When the mapped value is an atom, no parsarg is needed, else the necessary parsrgs are listed along with the case identifier (I,II etc)

```
1a --> I
1b --> II
.
.
3 --> (I xiye)
.
.
7e --> (IV moXye Weke)
.
.
```

fig. Section of the map for Bengali

The case identifier is the parameter on which the

oblique forms of a noun are indexed, i.e., given a noun root and a case marker, the relevant oblique noun form may be accessed. A declined form of a noun root is ,most generally, an oblique form of the root followed by one or more parsrgs- which are special words having no significance of their own.

4.2.2 TRIP RESUMED

(A) Using the TAM field in each vg, the TAM table is looked up and the verb-form found is installed in the vg. For our case, the value found is:

(inf (pAr pst-act-part))

The splitter creates a new entry for each aux. verb and adds them (in the order they appear in the verb-form) just after the vg. Information about person, number and gender is repeated.

(B)Using the vibh->case-parsrg map, if parsrg(s) are found for any ng, a new entry is created for each one and added just after the ng. Suppose an ng had vibhkt= 7e, since 7e maps to (IV moXye Weke), the splitter does the following:

```
(...(ng ....7e....)..) --> (...(n ...IV...)
                                (parsrg moXye)
                                (parsrg Weke) .....)
```

For our example, 1a -> I; 6 -> IV; 7b -> III, so the splitter output looks like:

```
[(adj (root ei) (positn 1) ... ((mod-adj-of **)))
 (n (root juwo) (positn 2) ...(case I).. ((krkrel (karwA-of **))))
 (n (root rAm) (positn 3) ....(case IV)...((mod-sam-of **)))
 (n (root pA) (positn 4) ....(case III)....((krkrel (xeSa-aXi-of **))))
 (adv (root Tik) (positn 5) .... ((mod-adv-of **)))
 (v (root ha) (positn 6) ....(verb-form inf).....)
 (v (root pAr) .....(verb-form pst-act-part).....)]
```

4.3 MORPHOLOGICAL SYNTHESIZER

This block accesses the lexicon , from which, using a word entry, the required word form may be looked up. For each word entry in the splitter output, a lexicon lookup is performed to output the wordform and thereby the target language sentence is generated.

Final output:

ei juwo rAmer pAe Tik howe pAre

4.3.1 STORAGE ISSUES

(a) verbs : the conjugated system of a verb is stored as an array which has the same name as the verb root and has the following indices:

(i) finite forms : verb-form, person ,number ,gender

(ii) infinite forms : verb-form

(b) nouns : the declined system of a noun is stored as an array which has the same name as the noun root and has the following indices: case identifier and number. Other lexical categories usually have no morphology.

chapter-5

CONCLUSIONS

5.1 SUMMING UP

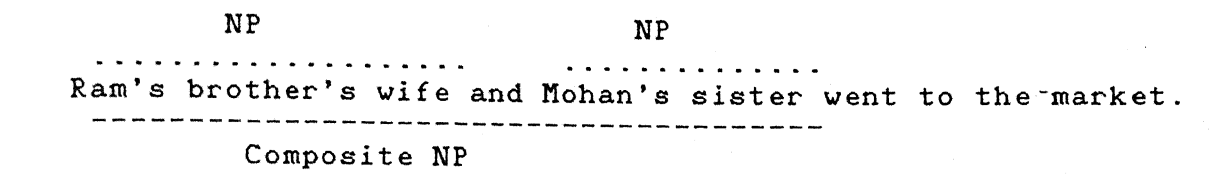
In this thesis, we have developed a language independent model of text synthesis in the context of machine translation using a transfer approach for handling language idiosyncracies and have tried to keep the model as close as possible to the functional converse of the analyzer component of the translator.

5.2 LIMITATIONS AND SCOPE FOR FUTURE WORK

The present version of the generator does not handle the following identified situations :

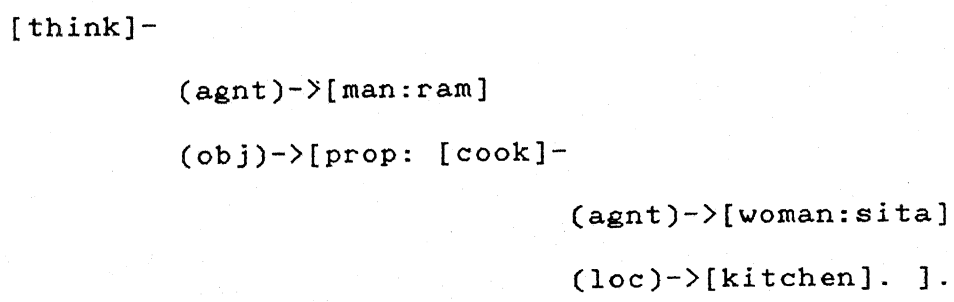
(i) Generating a composite noun phrase which is a conjunction, disjunction or both, of noun phrases :

for example,



This entails designing a device using which the language expert could easily express how a language handles conjunction and disjuction, i.e, how many times and where with respect to the NP's does it put the conjunctive and/or disjunctive markers. Integrating this device to the system would take care of the situation. This could also be extended to compound sentences, wherein the problem of pronominal substitution would come in.

(ii) Generating a concept with a graph referent: eg, Ram thinks Sita is cooking in the kitchen. The corresponding graph would be:



A trivial way to do this would be as follows:

for any concept with a graph referent, starting at

the innermost level, generate the sentence for the referent graph and put it in position with respect to the next outer level (surface ordering provided by the parser) with a sentence complementiser preceding it. This would hold for arbitrary levels of nesting but would probably not make very natural reading. There are possibly other and better ways of performing this task.

(iii) Handling incomplete data:

If relevant karak charts or noun charts are found to be missing at any stage, the system, at present, simply fails to function. An interactive mechanism for asking the user about the missing data ought to be incorporated.

APPENDIX-1

STRUCTURES FOR LEXICAL ENTRIES

(n root positn subcat gender number case
concpt sem-featur)

(v root positn subcat gender number person
caus-type tam cclusts)

(ng root positn subcat gender number vibhkt
concpt sem-featur)

(vg root positn subcat gender number person
caus-type tam cclusts)

SAMPLE RUNS

```
**********************************************************

[1](generate-sent CG8 'beng)


THE CONCEPTUAL GRAPH : [bETa-105:*a]-
        (caus-type)->[N:*a]
        (desh-adhi)->[kursi:*a]->(pos)->[num:4]
        (during)->[cunAv-ho:*a]-
                (caus-type)->[N:|*a#00007|]
                (pos)->[num:1],
        (karwA)->[ummixvAr:*a]-
                (mod-adj)->[xo:*a]->(pos)->[num:2]
                (pos)->[num:3],
        (pos)->[num:5]
        (tag)->[main:*a]
        (tam)->[yA_WA:*a].

**********************************************************

THE TRANSFER COMPONENT

THE TRANSFORMED CG:

[bETa-105:*a]-
        (caus-type)->[N:*a]
        (desh-adhi)->[kursi:*a]->(pos)->[num:4]
        (during)->[cunAv-ho:*a]-
                (caus-type)->[N:|*a#00007|]
                (pos)->[num:1],
        (karwA)->[ummixvAr:*a]-
                (mod-adj)->[xo:*a]->(pos)->[num:2]
                (pos)->[num:3],
        (pos)->[num:5]
        (tag)->[main:*a]
        (tam)->[yA_WA:*a].


**********************************************************
```

```
THE MORPH. SYNTHESIZER

THE SENTENCE :

nirbAcon hobAr Samay dujon prArWI ceyAre boSeCilo
(nirbachon hobar shamay dujon prarthi cheyare boshechilo)
**************************************************t

[2](generate-sent CG10 'beng)


THE CONCEPTUAL GRAPH : [KA-1:*a]-
        (caus-type)->[C:*a]
        (karana)->[Sahax:*a]->(pos)->[num:2]
        (karma)->[xavA:*a]->(pos)->[num:3]
        (karwA)->[baccA:{*}]->(pos)->[num:1]
        (pos)->[num:4]
        (tag)->[main:*a]
        (tam)->[yA_jA_sakwA_hE:*a].

**********************************************

THE TRANSFER COMPONENT


THE TRANSFORMED CG:

[KA-1:*a]-
        (caus-type)->[C:*a]
        (karana)->[Sahax:*a]->(pos)->[num:2]
        (karma)->[xavA:*a]->(pos)->[num:3]
        (karwA)->[baccA:{*}]->(pos)->[num:1]
        (pos)->[num:4]
        (tag)->[main:*a]
        (tam)->[yA_jA_sakwA_hE:*a].



******************************************************

THE MORPH. SYNTHESIZER

THE SENTENCE :

bAccAderke moXu diye oSuX KAvAno jewe pAre
(bachchaderke modhu diye oshudh khavano jeti pare)
**************************************************t
```

```
[3](generate-sent CG11 'beng)


THE CONCEPTUAL GRAPH : [bETa-102:*a]-
        (caus-type)->[N:*a]
        (desh-adhi)->[xil:*a]-
                (mod-sam)->[manuRya:rAm]->(pos)->[num:4]
                (pos)->[num:5],
        (karwA)->[dar:*a]->(pos)->[num:6]
        (pos)->[num:7]
        (shatri)->[KA-10a:*a]-
                (apadaan)->[aXyapak:*a]->(pos)->[num:1]
                (caus-type)->[N:|*a#00009|]
                (karma)->[mAr:*a]->(pos)->[num:2]
                (karwA)->[manuRya:|rAm#00008|]->(gen)->[no:*a]
                (pos)->[num:3],
        (tag)->[main:*a]
        (tam)->[|0_gayA|:*a].

**************************************************

THE TRANSFER COMPONENT

THE TRANSFORMED CG:

[KA-10a:*a]-
        (apadaan)->[aXyapak:*a]->(pos)->[num:1]
        (caus-type)->[N:|*a#00009|]
        (karma)->[mAr:*a]->(pos)->[num:2]
        (karwA)->[manuRya:|rAm#00008|]->(gen)->[no:*a]
        (pos)->[num:3]
        (shatri)<-[bETa-102:*a]-
                (caus-type)->[N:*a]
                (desh-adhi)->[xil:*a]-
                        (mod-sam)->[manuRya:rAm]->(pos)-
>[num:4]
                        (pos)->[num:5],
                (karwA)->[dar:*a]->(pos)->[num:6]
                (pos)->[num:7]
                (tag)->[main:*a]
                (tam)->[|0_gayA|:*a].


******************************************************

THE MORPH. SYNTHESIZER
```

THE SENTENCE :

oXyApaker kACe mAr Kewe Kewe rAmer mone Boy Duke gelo
(Codhyapaker kachhe mar khete khete ramer mone bhoy dhuke gelo)
**********t

A RUN SHOWING OUTPUTS AT EACH STAGE

[4](generate-sent CG1 'tel)

THE CONCEPTUAL GRAPH :

```
[KA-1:*a]-
        (apAxAna)->[aXikArI-1-2-3-4-5:*a]->(pos)->[num:2]
        (caus-type)->[N:*a]
        (karma)->[jUwA-1:*a]->(pos)->[num:3]
        (karwA)->[aparAXI-1-2:*a]->(pos)->[num:1]
        (pos)->[num:4]
        (tag)->[main:*a]
        (tam)->[|0_rahA_WA|:*a].
```

relation with current verb is tam

relation with current verb is tag

relation with current verb is pos

creating lexical bindings for the karwA

creating lexical bindings for the karma

relation with current verb is caus-type

creating lexical bindings for the apAxAna

****

THE TRANSFER COMPONENT

```
THE TRANSFORMED CG : [KA-1:*a]-
        (apAxAna)->[aXikArI-1-2-3-4-5:*a]->(pos)->[num:2]
        (caus-type)->[N:*a]
        (karma)->[mAra-1:*a]-
                (mod-sam)<-[jUwA-1:*a]->(pos)->[num:3]
                (pos)->[num:4];
```

```
        (karwA)->[aparAXI-1-2:*a]->(pos)->[num:1]
        (pos)->[num:5]
        (tag)->[main:*a]
        (tam)->[|0_rahA_WA|:*a].

**************************************************

THE GENERATOR

A LIST OF PHRASE GROUPS :

       ((ng aXikAri
              2
              nil
              n
              s
              nil
              aXikArI-1-2-3-4-5
              nil
              ((krkrel
                (apAxAna
                 (vg win
                      5
                      nil
                      m
                      s
                      a
                      N
                      |0_rahA_WA|
                      ((karwA m  1  (and  ($type  BOwika)  (not
($root  jaMga))))
                       (karma m 2 t)
                       (karana d 3 ($type BOwika_others))
                       (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                      nil)))
                 (person a)))
         (ng wannu
              4
              nil
              n
              s
              nil
              mAra-1
              nil
              ((mod-sam ((ng jodu 3 nil n  s  nil  jUwA-1  nil
```

```
((person a)))))
              (krkrel
               (karma
                (vg win
                     5
                     nil
                     m
                     s
                     a
                     N
                     |0_rahA_WA|
                     ((karwA m  1  (and  ($type  BOwika)  (not
($root jaMga))))
                      (karma m 2 t)
                      (karana d 3 ($type BOwika_others))
                      (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                     nil)))
               (person a)))
         (ng aparAXi
             1
             nil
             m
             s
             nil
             aparAXI-1-2
             nil
             ((krkrel
               (karwA
                (vg win
                     5
                     nil
                     m
                     s
                     a
                     N
                     |0_rahA_WA|
                     ((karwA m  1  (and  ($type  BOwika)  (not
($root jaMga))))
                      (karma m 2 t)
                      (karana d 3 ($type BOwika_others))
                      (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                     nil)))
               (person a)))
         (vg win
             5
             nil
```

```
             m
             s
             a
             N
             |0_rahA_WA|
             ((karwA m 1  (and  ($type  BOwika)  (not  ($root
jaMga))))
              (karma m 2 t)
              (karana d 3 ($type BOwika_others))
              (apAxAna  d  5  (or  ($type   manuRya)   ($type
sWAna))))
             nil))
**************************************************
STRUCTURE AFTER VIBHAKTI'S ARE PUT :

      ((ng aXikAri
             2
             nil
             n
             s
             5
             aXikArI-1-2-3-4-5
             nil
             ((krkrel
               (apAxAna
                (vg win
                    5
                    nil
                    m
                    s
                    a
                    N
                    |0_rahA_WA|
                    ((karwA m  1  (and  ($type  BOwika)  (not
($root jaMga))))
                     (karma m 2 t)
                     (karana d 3 ($type BOwika_others))
                     (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                    ((verb-type (duratv (un past)))))))
              (person a)))
        (ng wannu
             4
             nil
             n
             s
             2
             mAra-1
             nil
```

```
            ((mod-sam ((ng jodu 3 nil n  s  nil  jUwA-1     nil
((person a)))))
             (krkrel
              (karma
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)     (not
($root jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                   ((verb-type (duratv (un past)))))))
              (person a)))
         (ng aparAXi
             1
             nil
             m
             s
             1
             aparAXI-1-2
             nil
             ((krkrel
               (karwA
                (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)     (not
($root jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                   ((verb-type (duratv (un past)))))))
              (person a)))
         (vg win
             5
```

```
             nil
             m
             s
             a
             N
             |0_rahA_WA|
             ((karwA m 1  (and  ($type  BOwika)  (not  ($root
jaMga))))
              (karma m 2 t)
              (karana d 3 ($type BOwika_others))
              (apAxAna  d  5  (or  ($type   manuRya)   ($type
sWAna))))
             ((verb-type (duratv (un past))))))
```

STRUCTURE AFTER WORD GROUPS ARE BROUGHT TO THE SURFACE :

```
      ((ng aXikAri
             2
             nil
             n
             s
             5
             aXikArI-1-2-3-4-5
             nil
             ((mod-sam nil)
              (mod-adj nil)
              (krkrel
               (apAxAna
                (vg win
                    5
                    nil
                    m
                    s
                    a
                    N
                    |0_rahA_WA|
                    ((karwA m  1  (and  ($type  BOwika)  (not
($root  jaMga))))
                     (karma m 2 t)
                     (karana d 3 ($type BOwika_others))
                     (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                    ((verb-type (duratv (un past))))))
              (person a)))
        (ng jodu
            3
            nil
```

```
n
s
nil
jUwA-1
nil
((mod-sam nil)
 (mod-adj nil)
 (mod-sam-of
  (ng wannu
      4
      nil
      n
      s
      2
      mAra-1
      nil
      ((mod-adj nil)
       (mod-sam nil)
       (krkrel
        (karma
         (vg win
             5
             nil
             m
             s
             a
             N
             |0_rahA_WA|
             ((karwA m
                     1
                     (and ($type BOwika)
                          (not ($root jaMga))))
              (karma m 2 t)
              (karana d 3 ($type BOwika_others))
              (apAxAna d
                       5
                       (or    ($type    manuRya)
($type sWAna))))
                       ((verb-type (duratv (un past)))))))
                  (person a))))
   (person a)))
 (ng wannu
     4
     nil
     n
     s
     2
     mAra-1
```

```
            nil
            ((mod-adj nil)
             (mod-sam nil)
             (krkrel
              (karma
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)  (not
($root jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                   ((verb-type (duratv (un past)))))))
              (person a)))
        (ng aparAXi
            1
            nil
            m
            s
            1
            aparAXI-1-2
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (krkrel
              (karwA
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)  (not
($root jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                   ((verb-type (duratv (un past)))))))
```

```
          (person a)))
        (vg win
            5
            nil
            m
            s
            a
            N
            |0_rahA_WA|
            ((karwA m 1  (and  ($type  BOwika)  (not  ($root
jaMga))))
             (karma m 2 t)
             (karana d 3 ($type BOwika_others))
             (apAxAna  d  5  (or  ($type   manuRya)   ($type
sWAna))))
            ((verb-type (duratv (un past))))))

*****************************************************

STRUCTURE AFTER SORTING (PARSE STRUCTURE) :

      ((ng aparAXi
            1
            nil
            m
            s
            1
            aparAXI-1-2
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (krkrel
              (karwA
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)  (not
($root  jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                   ((verb-type (duratv (un past)))))))
            (person a)))
```

```
        (ng aXikAri
            2
            nil
            n
            s
            5
            aXikAri-1-2-3-4-5
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (krkrel
              (apAxAna
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)  (not
($root  jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                   ((verb-type (duratv (un past)))))))
              (person a)))
        (ng jodu
            3
            nil
            n
            s
            6
            jUwA-1
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (mod-sam-of
              (ng wannu
                  4
                  nil
                  n
                  s
                  2
                  mAra-1
                  nil
                  ((mod-adj nil)
```

```
                    (mod-sam nil)
                    (krkrel
                     (karma
                      (vg win
                          5
                          nil
                          m
                          s
                          a
                          N
                          |0_rahA_WA|
                          ((karwA m
                                  1
                                  (and ($type BOwika)
                                       (not ($root jaMga))))
                           (karma m 2 t)
                           (karana d 3 ($type BOwika_others))
                           (apAxAna d
                                    5
                                    (or    ($type    manuRya)
($type sWAna))))
                          ((verb-type (duratv (un past)))))))
                    (person a))))
              (person a)))
         (ng wannu
             4
             nil
             n
             s
             2
             mAra-1
             nil
             ((mod-adj nil)
              (mod-sam nil)
              (krkrel
               (karma
                (vg win
                    5
                    nil
                    m
                    s
                    a
                    N
                    |0_rahA_WA|
                    ((karwA m  1  (and  ($type  BOwika)  (not
($root jaMga))))
                     (karma m 2 t)
                     (karana d 3 ($type BOwika_others))
```

```
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                   ((verb-type (duratv (un past)))))))
             (person a)))
        (vg win
            5
            nil
            m
            s
            a
            N
            |0_rahA_WA|
            ((karwA m 1  (and  ($type  BOwika)  (not  ($root
jaMga))))
             (karma m 2 t)
             (karana d 3 ($type BOwika_others))
             (apAxAna  d  5  (or  ($type   manuRya)   ($type
sWAna))))
            ((verb-type (duratv (un past)))))))
```

THE SPLITTER STRUCTURE AFTER SPLITTING :

```
      ((ng aparAXi
            1
            nil
            m
            s
            1
            aparAXI-1-2
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (krkrel
              (karwA
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)  (not
($root  jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
```

```
                ((verb-type (duratv (un past)))))))
          (person a)))
        (ng aXikAri
            2
            nil
            n
            s
            5
            aXikAri-1-2-3-4-5
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (krkrel
              (apAxAna
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)  (not
($root  jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
aWAna))))
                   ((verb-type (duratv (un past)))))))
             (person a)))
        (ng jodu
            3
            nil
            n
            s
            6
            jUwA-1
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (mod-sam-of
              (ng vannu
                  4
                  nil
                  n
                  s
                  2
                  mAra-1
```

```
                    ((verb-type (duratv (un past)))))))
              (person a)))
        (ng aXikAri
            2
            nil
            n
            s
            5
            aXikArI-1-2-3-4-5
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (krkrel
              (apAxAna
               (vg win
                   5
                   nil
                   m
                   s
                   a
                   N
                   |0_rahA_WA|
                   ((karwA m  1  (and  ($type  BOwika)  (not
($root  jaMga))))
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sWAna))))
                  ((verb-type (duratv (un past)))))))
              (person a)))
        (ng jodu
            3
            nil
            n
            s
            6
            jUwA-1
            nil
            ((mod-sam nil)
             (mod-adj nil)
             (mod-sam-of
              (ng wannu
                  4
                  nil
                  n
                  s
                  2
                  mAra-1
```

```
                    nil
                    ((mod-adj nil)
                     (mod-sam nil)
                     (krkrel
                      (karma
                       (vg win
                           5
                           nil
                           m
                           s
                           a
                           N
                           |0_rahA_WA|
                           ((karwA m
                                   1
                                   (and ($type BOwika)
                                        (not ($root jaMga))))
                            (karma m 2 t)
                            (karana d 3 ($type BOwika_others))
                            (apAxAna d
                                     5
                                     (or    ($type    manuRya)
($type sWAna))))
                                  ((verb-type (duratv (un past))))))
                      (person a))))
               (person a)))
          (ng wannu
              4
              nil
              n
              s
              2
              mAra-1
              nil
              ((mod-adj nil)
               (mod-sam nil)
               (krkrel
                (karma
                 (vg win
                     5
                     nil
                     m
                     s
                     a
                     N
                     |0_rahA_WA|
                     ((karwA m  1  (and  ($type  BOwika)  (not
($root jaMga))))
```

```
                    (karma m 2 t)
                    (karana d 3 ($type BOwika_others))
                    (apAxAna d 5 (or ($type manuRya)  ($type
sUAna))))
                  ((verb-type (duratv (un past)))))))
            (person a)))
        (v win
           5
           nil
           m
           s
           a
           N
           |O_rahA_UA|
           ((karwA m  1  (and  ($type  BOwika)  (not  ($root
jaMga))))
            (karma m 2 t)
            (karana d 3 ($type BOwika_others))
            (apAxAna  d  5  (or   ($type   manuRya)   ($type
sUAna))))
           ((verb-type  duratv)  (verb-type  (duratv  (un
past)))))
        (v un  nil  nil  m  s  a  nil  nil  nil  ((verb-type
past))))

*********************************************************

THE MORPH. SYNTHESIZER


THE SENTENCE :

aparAXi aXikArinunci jodu wannuni wiMtU unnADu
(aparadhi adhikaranunchi jodu tannuni tintu unnadu)
*********************************************************

[5](generate-sent CG2 'tel)


THE CONCEPTUAL GRAPH :

[pA-2:*a]-
      (caus-type)->[N:*a]
      (karma)->[CadI-1:*a]->(pos)->[num:3]
      (karwA)->[cora-1-2-3-4-5:*a]->(pos)->[num:1]
      (pos)->[num:4]
      (tag)->[main:*a]
```

```
        (tam)->[yA:*a]
        (xeSa-aXi)->[almArI-1:*a]->(pos)->[num:2].

**********************************************************

THE TRANSFER COMPONENT

THE TRANSFORMED CG : [pA-2:*a]-
        (caus-type)->[N:*a]
        (karwA)->[CadI-1:*a]->(pos)->[num:3]
        (sampraxAna)->[cora-1-2-3-4-5:*a]->(pos)->[num:1]
        (pos)->[num:4]
        (tag)->[main:*a]
        (tam)->[yA:*a]
        (xeSa-aXi)->[almArI-1:*a]->(pos)->[num:2].


**********************************************************

THE MORPH. SYNTHESIZER


THE SENTENCE :

xoVMgaki almArulo juvva xoVrikiMxi

(dengaki almarulo juvva dorikindi)
**********************************************************
```

APPENDIX-2

SAMPLE RUN OF THE PARSER FOR THE EXAMPLE #4 IN APPENDIX-1

```
**********************************************************

THE MORPH. ANALYZER

(setq *sent* '(aparAXi aXikArI se jUwA KA rahA WA))
(setq *lex*
      '(((n aparAXI nil c m s d aparAXI-1-2 manuRya  ((nclusts  aparAXI-
1-2)))
        (n aparAXI nil c m s o aparAXI-1-2 manuRya  ((nclusts  aparAXI-
1-2)))
        (n aparAXI nil c m p d aparAXI-1-2 manuRya  ((nclusts  aparAXI-
1-2))))
       ((n  aXikArI  nil  c  m  s  d  aXikArI-1-2-3-4-5  nil  ((nclusts
aXikArI-1-2-3-4-5)))
        (n  aXikArI  nil  c  m  p  d  aXikArI-1-2-3-4-5  nil  ((nclusts
aXikArI-1-2-3-4-5)))
        (n  aXikArI  nil  c  m  s  o  aXikArI-1-2-3-4-5  nil  ((nclusts
aXikArI-1-2-3-4-5))))
       ((parsrg se nil any any any nil))
```

```
((n jUwA
    nil
    c
    m
    p
    d
    jUwA-1
    nil
    ((nclusts jUwA-1))))
((v KA nil v1a m s any n 0 (KA-1-2-3) nil)
 (v KA nil v1a m p any n 0 (KA-1-2-3) nil)
 (v KA nil v1a f s any n 0 (KA-1-2-3) nil)
 (v KA nil v1a f p any n 0 (KA-1-2-3) nil))
((v raha nil intransitive m s u normal yA (raha-kf) nil)
 (v raha nil intransitive m s m normal yA (raha-kf) nil)
 (v raha nil intransitive m s a normal yA (raha-kf) nil))
((v WA nil aux m s all normal all (hE-kf) nil))
((fullstop nil nil))
((fullstop nil nil))))
```

****

THE GROUPER

```
(setq *lwg*
```

```
'(((ng aparAXI
       1
       c
       m
       s
       ($enum 1 2)
       aparAXI-1-2
       manuRya
       ((parsrg d) (person nil) (nclusts aparAXI-1-2)))
  (ng aparAXI
       1
       c
       m
       p
       ($enum 1 2)
       aparAXI-1-2
       manuRya
       ((parsrg d) (person nil) (nclusts aparAXI-1-2))))
((ng aXikArI
       2
       c
       m
       s
       ($enum 3 5)
```

```
            aXikArI-1-2-3-4-5
            nil
            ((parsrg se) (person nil) (nclusts ...
        ((ng jUwA
            3
            c
            m
            p
            ($enum 1 2)
            jUwA-1
            nil
            ((parsrg d)
             (person nil)
             (nclusts jUwA-1))))
       ((vg KA 4 v1a m s any  |O_rahA_WA| (KA ; ; ;
       ((fullstop nil nil))
       ((fullstop nil nil))))
```

****

HE PARSER

HE CONCEPTUAL GRAPH :

```
KA-1:*a]-
      (apAxAna)->[aXikArI-1-2-3-4-5:*a]->(pos)
```

```
(caus-type)->[N:*a]

(karma)->[jUwA-1:*a]->(pos)->[num:3]

(karwA)->[aparAXI-1-2:*a]->(pos)->[num:1]

(pos)->[num:4]

(tag)->[main:*a]

(tam)->[|0_rahA_WA|:*a].
```

****

CONCEPTUAL GRAPHS.

Conceptual graphs are bipartite graphs. There are two types of nodes called as concepts and relations. The concepts represent entities,attributes,states and events and the relations represent the interlinking of these concepts. The mosaic of percepts that gets formed on the sensory excitation recieved by eyes, when a cat is sitting on a mat can be described by the following conceptual graph:

[cat]←(agent)←[sit]→(loc)→[mat].

The graph expresses, *"Some cat is agent of some action of sitting and the location of the act of sitting is some mat"*. In the linear notation of representing the conceptual graphs, the concepts are encased in square brackets while the relations are encased in parenthesis. The arcs which join the relations and the concepts are represented by arrows.

In general a concept has two fields, a *type* field and a *referent* field. In the above example *cat,mat* and *sit* are the types of three different concepts. The referent field of a concept is analogous to the variables used in symbolic logic. The referent can be of many types. A *generic referent* talks about an unspecified individual, analogous to the existantially quantified variables in symbolic logic. An instance of an individual is represented by an *individual referent*, which can be an identification number. A context is represented by a special type of concept: PROPOSITION, the referent of this concept can be a set of graphs. This set of graphs is said to occur in the context of the concept of type PROPOSITION.

A relation in general has *n* arcs. We call the relation as an *n-adic* relation. Each of the arcs of the relation is connected to a concept. Relation has only a type label. A relation in conceptual graphs can be viewed as an *n-adic* predicate in symbolic logic.

A hierarchy of concept types, which defines the relationships between concepts at different levels of generality, is called as the *type hierarchy*. It defines a partial ordering over the set of type labels. As an example the type

elephant can be called as a subtype of type *animal*, which can be represented by *elephant* $\leq$ *animal*.

Every possible combination of concept nodes and relation nodes which are interlinked may not make sense. For example following graph is perfectly possible but not sensible.

[sleep]→(agent)→[ideal]→(color)→[green].

To avoid such a nonsensical information, certain selectional constraints are essential on the permissible combination of concepts and relations. To distinguish the meaningful graphs that represent real or possible situations in the external world, certain graphs are declared to be *canonical*. New graphs may become canonical by perceptions, insights, or the formation rules. The way of deriving a new canonical graph from other canonical graphs by the rules of formation, is referred to as the process of *canonical derivation*.

The transfer component of the generator performs certain operations on conceptual graphs, in order to handle the idiosyncracies of a praticular language. These operation may not pertain to the selectional constraints. They are strictly for the purpose of modification of conceptual graphs for translation purpose. That is to say that, these operations do not obey the canonical rules of formation of conceptual graphs.

Conceptual graphs allow to define new types of concepts in form of a graph of more primitive types of concepts. This is parallel to the notion of explaining more complex actions or entities in terms of primitives, in any natural language. The relations also can be defined in terms of other graphs. This mechanism of is called as *type definition mechanism*.

MAPPING OF DEVNAGRI ALPHABETS

ONTO ROMAN ONES

a	A	i	I	u	U	q	e	E	eV
अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ऍ

o	O	OV	AN	AH
ओ	औ	ऑ	अं	अः

K	Kh	g	Gh	f	c	Ch	j	Jh	F
क़	ख़	ग़	घ़	ड़	च़	छ़	ज़	झ़	ञ़

t	T	d	D	N	w	W	x	X	n
ट़	ठ़	ड़	ढ़	ण़	त़	थ़	द़	ध़	ऩ

p	P	b	B	m	y	r	l	v	S
प़	फ़	ब़	भ़	म़	य़	ऱ	ल़	व़	श़

s	R	h
स़	ष़	ह़

GLOSSARY

Action Concept :

A concept that embodies an action, eg, KA-2, which embodies a certain class (captured by the number 2) of ingestion.

Argument Concept :

A concept that embodies an object or an entity which can be an argument of an action or may even modify an action or an entity, eg, jUwA-1, which embodies a certain class of footwear.

Akankshya:

This expresses the semantic demand of a word group for its arguments.

Karak:

These are the semantic roles which are filled up by the arguments of an action, eg,

karwA-- agent of the action

karma-- object of the action

karaNa-- instrument of the action

xeSa-aXikarana-- location of occurence of the action

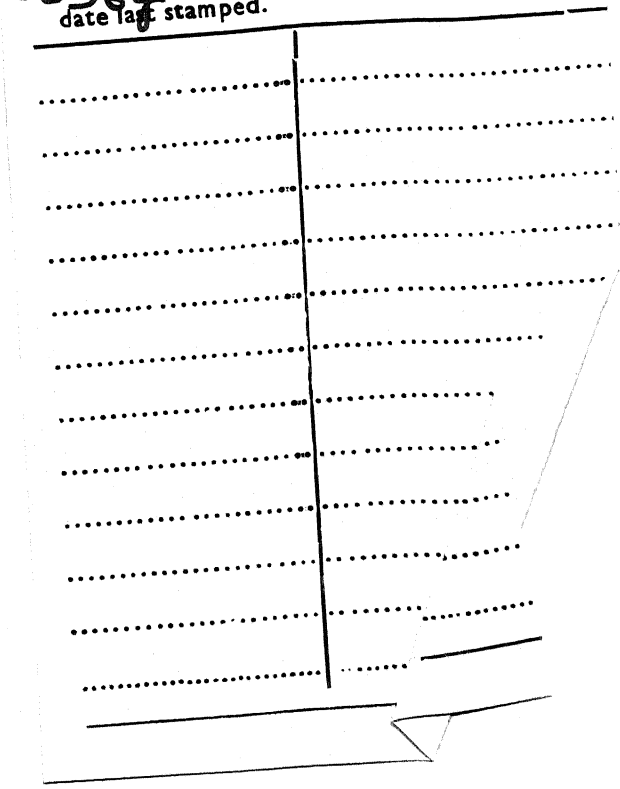
kAla-aXikarana-- time of occurence of the action.

Tam:

The tense, aspect and modality features of a verb form are rolled into one to compose the "tam" marker. Given a verb root and a tam marker, it is possible to access the corresponding conjugated verb form.

Yogyata:

This expresses the selectional constraints of a word group in order to satisfy the demands of another.



EE-1986-M-SEN-GEN